Parallel Rendering

(Draft Version for NSF Proposal)

1. Introduction

High-performance rendering of complex 2D and 3D data sets is important for interactive visualization and real-time immersive virtual environments. Typical architectural and mechanical CAD models contain millions of polygons [13], and scientific data visualization applications require rendering tens of millions of graphics primitives per second. High-performance rendering is particularly important for the display wall due to its high resolution, which enables display of very complex models, and its dimensions, which enable immersive applications requiring real-time refresh rates.

Traditionally, large displays have been driven by expensive rendering hardware, such as SGI's InfiniteReality Engine [14], which is able to render approximately ten million polygons per second at display resolutions up to a few million pixels. These rendering systems are expensive (typically over $100,000 each), and do not have adequate resolution to drive more than two projectors. To construct a large-scale system, it would require multiple rendering pipelines. In contrast, high-end PC-based graphics accelerator cards are capable of rendering several millions of polygons per second and filling about 60M textured pixels per second for less than $3,000. The price/performance ratio of these commodity graphics accelerators is about 20 times better than high-end SGI systems, and it is improving each year [15]. Based on these observations, it is compelling to consider parallel approaches that combine many PCs with commodity graphics accelerators together into a high performance, high resolution, and low-cost parallel rendering system.

2. Rationale

The goal of this project is to develop a system for real-time display of very large, high-resolution images partitioned over multiple projectors using multiple graphics accelerators in PCs connected by a system area network. As compared to previous parallel rendering research, this goal has several unique characteristics that present potential advantages and interesting research challenges.

First, we use commodity hardware components. As a result, we expect our system to be less expensive, more flexible, and track technology better than systems with custom hardware, and it should be less expensive and higher performance than systems based only on software. Of course, in using off-the-shelf components, we are constrained by their limitations. For instance, if we assume that geometry processing and rasterization are tightly coupled within the graphics accelerator, as is the case with most available cards, it may be impractical to employ a sort-middle approach to parallel rendering in which graphics primitives are repartitioned after geometry processing and before rasterization [21]. Instead, we must develop sort-first or sort-last parallel algorithms that access the commodity graphics cards only via standard APIs, such as OpenGL [22], and via direct access to the frame buffer.

Second, we use a system area network for communication among processors. As compared to hardware or shared memory multiprocessors, the advantages of networked clusters are modularity and flexibility. Since the processors of our system communicate only by network protocols, they may be added and removed from the system easily, and the system may be comprised of heterogeneous computers. Also, specific rendering processors can be accessed directly by remote computers attached to the network, and the communication capacity of our system scales at the rate of system area network improvements. The challenge of cluster computing is to overcome the bandwidth and latency characteristics of network communication. Previous parallel rendering methods, based on sort-middle and fine-grained subdivision of image space into many small tiles, have not performed well with a network of workstations, since the communication costs of low-bandwidth networks, operating system overheads, and heavyweight protocols thwart any potential speedups due to parallelism. The challenge is to develop fast, low-overhead methods for network communication and to choose a coarse-grained partitioning scheme, which scales within the bandwidth and latency limitations of the network.

Third, we render images for display on a very large surface. Unlike a typical desktop graphics display, for which one can assume a single user is viewing the entire screen from a known viewpoint (directly in front of the screen), users may walk around in front of the display wall and look at small parts of it from widely varying viewpoints. In order to avoid distortions due to mismatched 3D perspectives, and to take advantage of localized focii of attention, we must track user viewpoints and alter the rendering parameters used to refresh different regions of the screen according to the vantage point(s) of relevant user(s) and properties of the scene in that region.

Finally, we display an image composed of multiple sub-images corresponding to projectors attached to frame buffers of different PCs. This characteristic provides a natural image-parallel decomposition of the rendering computation. If we sort the graphics primitives based on their overlaps with the projection regions on the wall, then each PC must render only the subset of primitives overlapping its projector's region, and speedups due to parallelism might be achieved. In this case, composition of the sub-images rendered separately on every PC is handled by the projectors, without image processing or network communication. However, if the graphics primitives are not uniformly distributed over the screen (e.g., they may all reside in the region of one projector), or if we have available more graphics processors than there are projectors, then this simple, static partitioning approach does not achieve optimal performance. We must develop algorithms that dynamically partition the graphics primitives among processors so that the rendering performance is balanced and scales with increasing numbers of PCs and projectors.

3. Approach

Our approach is to partition the rendering computation into coarse-grained tasks requiring little inter-process communication using "virtual tiles," non-overlapping regions of the screen not necessarily corresponding one-to-one with projection regions (physical tiles). Unlike previous systems [23, 24], we allow virtual tiles to be any size or shape, including non-rectilinear, concave, or even disjoint regions, as long as each pixel on the screen maps to exactly one virtual tile. We use virtual tiles to sort graphics primitives among processors during parallel rendering. Specifically, each virtual tile is assigned to a processor, and each processor renders an image containing all graphics primitives that at least partially overlap its virtual tile. Since virtual tiles do not overlap, pixels of images rendered independently for separate virtual tiles can be combined without depth comparisons to construct images for projection on the wall. Pixels of a virtual tile rendered on one computer, but projected onto the wall by another, are sent over a system area network and loaded into the projected frame buffer for display each frame, as shown in Figure 2(a).

(a) Compositing virtual tiles. (b) Scene adaptive virtual tiles.

Figure 1: Virtual Tiling.

A primary advantage of using virtual tiles is that they provide a coarse-grained, data parallel decomposition of the rendering computation. Most previous parallel rendering systems have partitioned image space into R*P small, rectangular tiles (R=20) and then dynamically scheduled rendering of these tiles across the P processors to balance the load [25, 26]. That approach requires a large amount of fine-grained interprocess communication, which can lead to poor speedup results on PCs connected by a commercial system area network. Moreover, smaller tiles lead to larger primitive-tile overlap factors and less screen space coherence which further thwart speedups due to parallelism. We plan to construct exactly P virtual tiles that can be rendered independently on P processors. Interprocess communication will occur only to transfer pixels rendered for a virtual tile rendered on one computer to the frame buffer of another computer attached to the corresponding projector. For these transfers, we plan to implement enhanced network interfaces (described in Section XXX) supporting DMA between frame buffers and the Myrinet network [7, 8] with about 100 Mbytes/sec bandwidth and about 10 microseconds latency.

A second advantage of virtual tiles is that they may be dynamically resized and reshaped based on changing rendering conditions. Of course, the system can adapt the virtual tiling dynamically in real-time in order to balance the rendering load among processors based on measured and predicted scene complexity in different regions of the screen. However, dynamic rendering with virtual tiles provides several interesting opportunities beyond load balancing. Since each virtual tile is rendered independently on a separate graphics accelerator, it is possible to render different virtual tiles with different frame rates, different resolutions, and different viewpoints. For instance, we may adjust the view perspectives used to render different virtual tiles to match the viewpoint(s) of the user(s) looking at them. Or, we may render some virtual tiles at a faster frame rate or higher resolution than the others, because a user is looking at them, or because an object is moving rapidly across them.

A third advantage of virtual tiles is that they are arbitrary shapes, and thus they can better match the natural boundaries of scene geometry. This is important because it is possible to use irregular 2D projections of 3D multi-resolution data structures encoding the spatial distribution of graphics primitives to infer suitable virtual tile boundaries. Moreover, these same multi-resolution data structures can be used in a sort-first architecture to distribute graphics primitives among virtual tiles efficiently via a top-down traversal that avoids checking every primitive independently [27]. In previous work, we have used spatial subdivision data structures based on BSP trees [28] and bounding volume hierarchies [29] for real-time visibility determination and detail elision [30]. We plan to augment these data structures with scene complexity information to compute suitable virtual tiling for complex 3D scenes and to sort graphics primitives among virtual tiles in real-time. For instance, Figure 2(b) shows an image partitioned into four convex virtual tiles of roughly equal scene complexity. In this case, large groups of polygons (e.g., the 1,800 polygons representing each bookcase) can be sorted among virtual tiles with a few simple 2D intersection checks between the virtual tile regions and the projections of 3D hierarchical bounding volumes. Irregular virtual tile shapes derived these spatial data structures are also effective at separating complex regions into separate tiles and minimizing overlaps between primitives and tiles.

The challenge is to develop algorithms that compute the shapes and arrangement of virtual tiles dynamically and sort graphics primitives among virtual tiles in real-time. An effective virtual tiling must balance many conflicting factors. First, it should partition the screen into tiles that match regions requiring different view parameters, refresh rates, and resolutions. Second, it should partition graphics primitives evenly among tiles so as to balance the rendering load among processors. Third, it must enable rapid sorting of 3D graphics primitives among tiles in order to avoid undue processing overhead during primitive sorting. Fourth, since geometry processing must be performed redundantly for each primitive overlapping more than one virtual tile, the virtual tiling should be constructed so as to minimize overlaps between graphics primitives and tiles. Finally, if any rendering processor resides upon the same machine as a frame buffer attached to a projector, it is advantageous to match the virtual tile(s) processed by that processor with the physical tile corresponding to its frame buffer in order to avoid communication between rendering processors and frame buffers.

We plan to develop algorithms that balance these many factors to adjust the shape, size, and rendering attributes of each virtual tile dynamically based on statistics regarding: 1) predicted properties of the scene, 2) tracked user viewing information, and 3) measured rendering rates. Specifically, a master process monitors the distribution and motion of graphics primitives across the screen using dynamically updated hierarchical bounding volumes based on BSP trees augmented with scene complexity statistics. Periodically, it receives messages identifying regions of the screen of special interest to different users from separate processes tracking the users' viewpoints and focii with cameras. Also, it receives small, infrequent messages from rendering process characterizing its measured rendering load. Asynchronously, at suitable intervals, the master process uses the statistics it has gathered to adjust the virtual tiling, to select viewing parameters for each virtual tile, and to assign virtual tiles to rendering processors. In this manner, we plan to develop a sort-first system which adapts the partition of 3D graphics primitives among processors in real-time and uses the full rendering capabilities of the commodity graphics accelerator card in each PC.

Initially, we plan to use virtual tiles only to balance the load among rendering processors. In the first implementation, all virtual tiles will be rendered with the same viewing parameters, frame rates, and resolutions. Also, every graphics primitive will be replicated on each processor. In a second version, we plan to incorporate statistics regarding object motion and user viewpoints to guide computation of virtual tilling that may be rendered with different viewing parameters, frame rates, and resolutions. For this implementation, we will investigate methods to blend across the boundaries between virtual tiles rendered with different view parameters, and we will develop efficient methods to use texture-mapping hardware to resample images rendered at different resolutions before display. In a third version, we plan to investigate methods for migration of graphics primitives among processors.

4. Related Work

Previous parallel systems for polygon rendering can be classified in many ways: hardware vs. software, shared memory vs. distributed memory, object decomposition vs. image decomposition, SIMD vs. MIMD, sort-first vs. sort-middle vs. sort-last, or functional vs. data vs. temporal partitioning. See [21, 31-33] for more on classifications of parallel rendering approaches.

Hardware-based systems employ custom integrated circuits with fast, dedicated interconnections to achieve high processing and communication bandwidths. Early parallel rendering hardware was developed for z-buffer scan conversion [34-36] and for geometry processing pipelining [37, 38]. Most current systems are based on a sort-middle architecture and rely upon a fast, global interconnection to distribute primitives from geometry processors to rasterizers. For instance, UNC's Pixel Planes 5 [26] utilizes a fast, multi-channel token ring to connect the output of geometry processing chips to the input of custom CMOS VLSI processor-enhanced memories. Similarly, SGI's InfiniteReality Engine [19] uses a fast Vertex Bus to broadcast screen space vertex information from semi-custom ASIC geometry processors to ASIC rasterization processors. UNC's PixelFlow [39, 40] is based on a sort-last architecture and uses daisy-chained logic-enhanced memories to implement an image composition network that combines partial, full-screen images rendered independently by separate processors.

Software-based systems have been implemented on massively parallel architectures, such as Thinking Machines' CM-5 [41], BBN's Butterfly TC2000 [42], and Intel's Touchstone Delta system [25], and on shared memory multiprocessors, such as SGI's PowerChallenge [43]. Although shared memory facilitates inter-process communication and avoids partitioning and duplication of graphical data structures, these systems may not scale to a large number of processors for real-time applications. Also, general-purpose processors used in purely software systems are not optimized for rendering operations, nor are they directly connected to video display hardware, and thus their cost/performance ratio is generally not competitive with hardware implementations.

Relatively little work has been done on interactive polygon rendering using a network of PCs [44, 45]. The primary problem is that local area networks do not provide enough latency and bandwidth for scalable graphics performance. For instance, composition of images for display at interactive frame rates is problematic if each image is partitioned among multiple workstations during rendering. To address this issue, Wei et al. [46-48] have developed a multi-ported, network-attachable frame buffer. Prior parallel rendering systems have focused on inter-frame parallelism [44], rendering separate frames of a sequence on separate computers in parallel, rather than on intra-frame parallelism, which is required to achieve scalable speedups for numerous workstations. Networks of workstations have been used successfully for parallel algorithms with coarse granularity, such as volume rendering [49, 50], ray tracing {***}, radiosity [10, 51], and batch rendering of image sequences [44, 52]. We are not aware of any prior system that has achieved scalable polygon rendering speedups via intra-frame data parallelism across a network of workstations.

Many parallel rendering systems partition image-space into tiles for concurrent rasterization. Different tile shapes have been used, including scan lines [53], horizontal strips [23, 32, 54], vertical strips [23], and rectangular areas [42, 54, 55]. Tiles have been kept static [19, 25, 36, 54] or dynamically adjusted based on the distribution of graphics primitives on the screen [23, 24, 55]. Most tiled systems define several times more tiles than there are processors so that the rendering load can be balanced [25, 26]. This approach leads to inefficiencies due to high primitive-tile overlap factors and the loss of spatial coherence across tile boundaries [56]. Whitman uses fewer initial tiles and splits them adaptively in a task stealing approach [42]. To our knowledge, no prior system has considered using non-rectangular tiles or 3D spatial data structures encoding the distribution of scene complexity to guide dynamic adaptation of tiles. Also, we are not aware of any prior system that has used tiles to vary rendering parameters in different regions across the screen.

5. Related Work

[1] T. Mayer, “New Options and Considerations for Creating Enhanced Viewing Experiences,” *Computer Graphics,* 31(2): 32-34, May 1997.

[2] S. Shostak, *Advanced Imaging*, August 1992.

[3] PowerWall, http://lcse.umn.edu/powerwall/powerwall.html

[4] M. Czernuszenko, D. Pape, D. Sandin, and T. DeFanti, “The ImmersaDesk and InfinityWall Projection-Based Virtual Reality Displays,” *Computer Graphics,* 31(2): 46-52, May 1997.

[5] J. M. Younse, “Projection Display Systems Based on the Digital Micromirror Device (DMD),” in *Micromachining and Microfabrication '95 Conference,* October 1995.

[6] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su, “Myrinet: A Gigabig-per-Second Local Area Network,” *IEEE MICRO,* 15(1): 29--36, February 1995.

[7] C. Dubnicki, A. Bilas, K. Li, and J. Philbin, “Design and Implementation of Virtual Memory-Mapped Communication on Myrinet,” in *Proceedings of the IEEE 11th International Parallel Processing Symposium*, April 1997.

[8] Argus, http://www-graphics.stanford.edu/projects/flashg/argus/

[9] T. Funkhouser, C. Sequin, and S. Teller, “Management of Large Amounts of Data in Interactive Building Walkthroughs,” in *ACM SIGGRAPH Special Issue on 1992 Symposium on Interactive 3-D Graphics, Computer Graphics*, pages 11-20, March 1992.

[10] R. Cen and J. P. Ostriker, “CDM Cosmology with Hydrodynamics and Galaxy Formation: The Evolution of the IGM and Background Radiation Fields,” *Astrophysical Journal*, 1993.

[11] R. Cen, J. Miralda-Escude, J. P. Ostriker, and M. Rauch, “Gravitational Collapse of Small-Scale Structure as the Origin of the Lyman Alpha Forest,” *Astrophysical Journal Letters,* 437(L9), 1994.

[12] A. Finkelstein, C. E. Jacobs, and D. H. Salesin, “Multiresolution Video,” in *ACM SIGGRAPH 96 Conference Proceedings*, pages 281-290, 1996.

[13] E. Brechner, “Interactive Walkthroughs of Large Geometric Databases,” in *Course Notes, SIGGRAPH 96*, 1996.

[14] J. S. Montrym, D. R. Baum, D. L. Dignam, and C. J. Migdal, “InfiniteReality: A Real-Time Graphics System,” in *Computer Graphics (SIGGRAPH 97)*, pages 293-303, August 1997.

[15] SPEC, http://www.spec.org/gpc/

[16] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs, “A Sorting Classification of Parallel Rendering,” *IEEE Computer Graphics and Applications,* 14(4): 23-32, July 1994.

[17] *OpenGL Reference Manual, version 1.1*. OpenGL Architecture Board. Addison-Wesley, 1997.

[18] D. Ellsworth, “A New Algorithm for Interactive Graphics on Multicomputers,” *IEEE Computer Graphics and Applications,* 14(4): 33-40, July 1994.

[19] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebss, and L. Israel, “Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories,” in *Computer Graphics (SIGGRAPH 89)*, pages 79-88, July 1989.

[20] C. Mueller, “Hierarchical Graphics Databases in Sort-First,” in *Proceedings of the IEEE Symposium on Parallel rendering*, pages 49-57, 1997.

[21] H. Fuchs, Z. Kedem, and B. Naylor, “On Visible Surface Generation by a Priori Tree Structures,” in *Computer Graphics (SIGGRAPH '80)*, pages 124-133, 1980.

[22] J. Clark, “Hierarchical Geometric Models for Visible Surface Algorithms,” *Communications of the ACM,* 19(10): 547-554, October, 1976.

[23] T. A. Funkhouser, “Coarse-Grained Parallelism for Hierarchical Radiosity Using Group Iterative Methods,” in *Computer Graphics (SIGGRAPH 96)*, August 1996.

[24] S. Whitman, *Multiprocessor Methods for Computer Graphics Rendering*. AK Peters, 1992.

[25] T. W. Crockett and T. Orloff, “A MIMD Rendering Algorithm for Distributed Memory Architectures,” in *ACM SIGGRAPH Symposium on Parallel Rendering*, T. Crockett, C. Hansen, and S. Whitman, Eds. ACM Press, New York, NY, pages. 35--42, November 1993.

[26] T. W. Crockett, “An Introduction to Parallel Rendering,” *Parallel Computing,* 23: 819-843, 1997.

[27] H. Fuchs, “Distributing a Visible Surface Algorithm over Multiple Processors,” in *Proceedings of the 1977 ACM Annual Conference*, Seattle, WA, pages 449-451, 1977.

[28] H. Fuchs, "An Expandable Multiprocessor Architecture for Video Graphics," in *Proceedings of the 6th Annual ACM-IEEE Symposium on Computer Architecture*, pages 58-67, April 1979.

[29] F. I. Parke, "Simulation and Expected Performance Analysis of Multiple Processor Z-Buffer Systems," in *Computer Graphics (SIGGRAPH 80)*, pages 48-56, July 1980.

[30] J. Clark, "A VLSI Geometry Processor for Graphics," *IEEE Computer*, 13(7): 59-68, July 1980.

[31] J. Clark, "The Geometry Engine: A VLSI Geometry System for Graphics," in *Computer Graphics (SIGGRAPH 82)*, pages 127-133, July 1982.

[32] S. Molnar, J. Eyles, and J. Poulton, "PixelFlow: High-speed Rendering using Image Composition," *Computer Graphics*, 26(2): 231--240, 1992.

[33] J. Eyles, S. Molnar, J. Poulton, T. Greer, A. Lastra, N. England, and L. Westover, "PixelFlow: The Realization," in *Proceedings of the 1997 Siggraph/Eurographics Workshop on Graphics Hardware*, Los Angeles, CA, pages 57-68, August, 1997.

[34] F. A. Ortega, C. D. Hansen, and J. P. Ahrens, "Fast Data Parallel Polygon Rendering," in *Proceedings of Supercomputing '93*, IEEE Computer Society Press, pages 709-718, November 1993.

[35] S. Whitman, "Dynamic Load Balancing for Parallel Polygon Rendering," *IEEE Computer Graphics and Applications*, 14(4): 41-48, July, 1994.

[36] J. P. Singh, A. Gupta, and M. Levoy, "Parallel Visualization Algorithms and their Implications for Multiprocessor Architecture," *IEEE Computer*, 27(6), June 1994.

[37] J. Hubbell, M. Kaplan, and D. P. Greenburg, "Parallel Processing Techniques for Hidden Surface Removal," *Network Rendering, Autodesk University Sourcebook, Vol. 2*, 13(2): 443-453, 1996 July, 1979.

[38] G. Stoll, B. Wei, D. Clark, E. Felten, and K. Li, "Evaluating Multi-Port Frame Buffer Designs for a Mesh-Connected Multicomputer," in *Proceedings of the 22nd Annual Symposium on Computer Architecture*, pages 96-105, May 1995.

[39] B. Wei, G. Stoll, D. Clark, E. Felten, and K. Li, "Synchronization for a Multi-Port Frame Buffer on a Mesh-Connected Multicomputer," in *IEEE Parallel Rendering Symposium '95*, Oct 1995.

[40] B. Wei, D. Clark, E. Felten, K. Li, and G. Stoll, "Performance Issues of a Distributed Frame Buffer on a Multicomputer," in *The 1998 Eurographics/SIGGRAPH Workshop on Graphics Hardware*, Lisbon, Portugal, 1998.

[41] C. Grietsen and J. Petersen, "Parallel Volume Rendering on a Network of Workstations," *IEEE Computer Graphics and Applications*, 13(6): 16-23, November 1993.

[42] K. L. Ma, J. S. Painter, C. D. Hansen, and M. F. Krogh, "Parallel Volume Rendering Using Binary-Swap Compositing," *IEEE Computer Graphics and Applications*, 14(4): 59-68, July 1994.

[43] R. J. Recker, D. W. George, and D. P. Greenberg, "Acceleration Techniques of Progressive Refinement Radiosity," in *Computer Graphics (Proceedings of the 1990 Symposium on Interactive 3D Graphics)*, pages 59-66, March 1990.

[44] S. Upstill, *The RenderMan Companion*. Addison-Wesley, 1990.

[45] P. N. Glaskowsky, "Advanced 3D chips show promise," *Microprocessor Report*, 11(8): 5-9, June, 1997.

[46] M. Kaplan and D. P. Greenberg, "Parallel Processing Techniques for Hidden Surface Removal," in *Computer Graphics (SIGGRAPH '79 Proceedings)*, pages 300--307, August 1979.

[47] D. S. Whelan, "Animac: A Multiprocessor Architecture for Real-time Computer Animation," Ph.D. thesis, California Institute of Technology, Pasadena, CA 1985.

[48] D. R. Roble, "A Load Balanced Parallel Scanline Z-Buffer Algorithm for the iPSC Hypercube," in *Procedings of PIXIM '88*, Hermes, Paris, France, pages 177-192, October 1988.

[49] C. Mueller, "The Sort-First Rendering Architecture for High-Performance Graphics," in *ACM SIGGRAPH Computer Graphics (Special Issue on 1995 Symposium on Interactive 3-D Graphics)*, April 1995.

[50] M. Cox and N. Bhandari, "Architectural Implications of Hardware-Accelerated Bucket Rendering on the PC," in *Proceedings of the 1997 Siggraph/Eurographics Workshop on Graphics Hardware*, Los Angeles, CA, pages 25-34, 1997.

[51] S. M. Seitz and C. R. Dyer, "View Morphing: Synthesizing 3D Metamorphoses Using Image Transforms," in *ACM SIGGRAPH 96 Conference Proceedings, Annual Conference Series*, H. Rushmeier, Ed., pages. 21-30, August 1996.

[52] J. K. Aggarwal and Q. Cai, "Human Motion Analysis: A review," in *Proceedings of IEEE Computer Society Workshop on Motion of Non-Rigid and Articulated Objects*, pages 90-102, 1997.

[53] K. Sato, T. Maeda, H. Kato, and S. Inokuchi, "CAD-based object tracking with distributed monocular camera for security monitoring," in *Proceedings of the 2nd CAD-Based Vision Workshop*, pages 291-297, February 1994.

[54] I. A. Kakadiaris and D. Metaxas, "3D human body model acquisition from multiple views," in *Proceedings of the 5th International Conference on Computer Vision*, pages 618-623, 1995.

[55] S. E. Chen, "Quicktime VR - An Image-Based Approach to Virtual Environment Navigation," in *ACM SIGGRAPH 95 Conference Proceedings*, pages 29--38, August 1995.

[56] Y. Horry, K. Anjyo, and K. Arai, "Tour Into the Picture: Using a Spidery Mesh Interface to Make Animation from a Single Image," in *ACM SIGGRAPH 97 Conference Proceedings*, pages 225-232, August 1997.

[57] P. E. Debevec, C. J. Taylor, and J. Malik, "Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach," in *ACM SIGGRAPH 96 Conference Proceedings*, pages 11-20, August 1996.

[58] S. Coorg, N. Master, and S. Teller, "Acquisition of a Large Pose-Mosaic Dataset," in *Proceedings of the 1998 IEEE conference CVPR, to appear*, June 1998.

[59] F. Pighin, J. Hecker, D. Lischinski, R. Szeliski, and D. Salesin, "Synthesizing Realistic Facial Expressions from Photographs," in *ACM SIGGRAPH 98 Proceedings, to appear*, July 1998.